第3章

Bayerパターンの生データを使って効率良く画像処理を行う

――デジタル・カメラの“もったいない”処理を改善するテクニック

外村元伸

画像の拡大・縮小スケーリング表示などに使われる補間法の一つに，バイキュービック・スプライン法があります．補間精度の高い方法なのですが，例えばデジタル・カメラの画像を拡大・縮小スケーリング表示する場合，画像入力の段階でカラー・イメージ・センサ特有の問題があり，バイキュービック・スプライン補間法の能力を十分に発揮できません．本稿では，この問題がイメージ・センサのBayerパターンと呼ばれているカラー・フィルタ配列に起因していることを説明します．また，最近のデジタル・カメラでサポートされているカラー・フィルタ配列画像の生データを取り出す機能を利用して，その問題を解決する手段を紹介します．（筆者）

ついに画素数1,000万クラスのデジタル・カメラが登場し，今では比較的購入しやすい価格で発売されています．現在，R（赤）G（緑）B（青）の3原色カラー画像の取得と画像データの圧縮・保存は当たり前ですが，どのようにしてカラー画像を取得しているのか，皆さんご存じでしょうか．

1 画像の取得とBayerパターン

ところで，デジタル・カメラで標準的にサポートされている圧縮保存形式のJPEGは，せっかく取得した画像の細部の情報の一部を切り捨ててしまっています．どのように回復処理を行おうとも，切り捨てた情報は正確には戻りません．そのためJPEGのような画像圧縮方式は，非可逆圧縮と呼ばれています．

● 単板カラー・イメージ・センサの仕組み

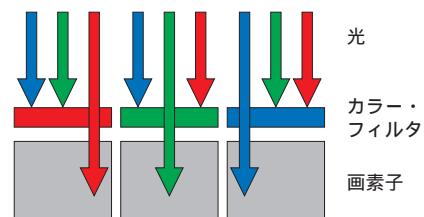
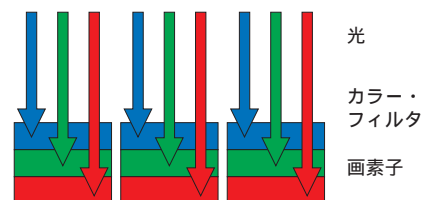
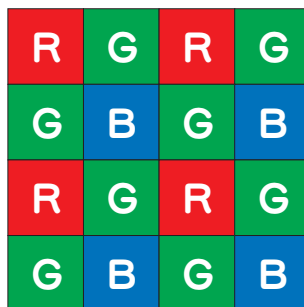
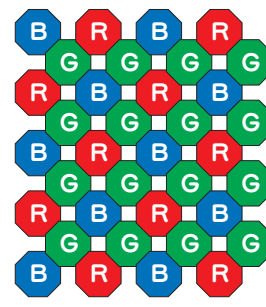
さらに，通常のデジタル・カメラでは，圧縮保存する前に既に取得した画像とは異なる画像に変換しています．これはカラー画像の取得方法に問題があるからです．単板のイメージ・センサ（撮像素子）からは，グレイスケールの画素値しか得られません．カラー画像を取得するには，図1（a）に示すように画素の前面にRGB3色のカラー・フィルタを取り付ける必要があります．民生機器では，コストと処理の複雑さを軽減するため，単板イメージ・センサを採用しています．単板では一つの画素位置に1色分のカラ

（a） Bayerパターン・カラー・フィルタのイメージ・センサ

（b） Foveon社のイメージ・センサ

図1　イメージ・センサの構造

（a）にBayerパターン・カラー・フィルタの構造を，（b）に米国Foveon社のイメージ・センサの構造を示す．

KeyWord　Bayerパターン，デジタル・カメラ，JPEG，非可逆圧縮，イメージ・センサ，グレイスケール，ハニカム構造，デモザイキング，バイキュービック・スプライン，バイリニア，2次元Wavelet変換

図2
Bayerパターン・カラー・フィルタの配置パターンの例
(a)や(b)の配置パターンがよく使われている．(c)は富士フイルムが採用している配置パターンで，「ハニカム(honeycomb)構造」と呼ばれる．

(**a**) 配置パターン1

(**b**) 配置パターン2

(**c**) ハニカム構造

ー・フィルタしか取り付けることができません．ただし，例外として米国Foveon社のイメージ・センサが挙げられます(**図1(b)**)．同社のイメージ・センサは，Si(シリコン)が深さ方向で異なる色の光を吸収することを利用し，同位置の深さ方向に3層を設けることでRGBを同時に取り込むことができます[注1]．

さて，一般の(Foveon社以外の)単板イメージ・センサでは，平面層に均等な面積で画素を並べるとすると，4(＝2×2)画素に対して3色のカラー・フィルタを配置することになります．そのため，どうしてもある1色に対して2画素を割り当てることになります．人間の目にとって一番感度が良く，知覚される最も重要な情報を含んでいる色が中間波長のG(緑)であることから，Gを2画素分割り当てるのが普通です．このようなカラー・フィルタの配列は，米国Eastman Kodak社のBryce E. Bayer氏が基本特許として出願し，米国特許3,971,065(July 20，1976)として登録されているため，今日では「Bayerパターン・カラー・フィルタ配列(Bayer Pattern Color Filter Array)」と呼ばれています．

● メーカごとに異なるBayerパターンの配置と構造

Bayerパターン・カラー・フィルタ配列の配置として，**図2(a)**，**(b)**のようなものが多く使われていますが，細かい配列パターンはデジタル・カメラ・メーカごとに異なります．また同じメーカでも機種間で異なるケースがあります．なお，**図2(c)**は富士フイルムが独自に採用している配置です．画素の形状を八角形(六角形ではないことに注意)にしてハチの巣状に並べていることから，「ハニカム(honeycomb)構造」と呼ばれています．

本稿では，Bayerパターン・カラー・フィルタ配列の典型的な例としてキヤノンの「EOS Kissデジタル X」を，ハニカム構造の例として富士フイルムの「FinePix S5200[注2]」を取り上げます．メーカが公開していない情報に関しては，筆者が独自に解析したものであり，保証するものではありません．読者の皆さんの責任において参考にしてください．また，ここで取り上げる以外の機種については，同じ機種系列であれば画素数の違い程度と考えられるので，これも本稿を参考にして，詳しくは皆さんで調べてみてください．

● 画像データ保存量はフルカラーRGBの1/3

図2から明らかなように，Bayerパターン・カラー・フィルタ配列を用いると，一つの画素位置にはRGB3色のうちの1色分の画素データ(グレイスケール)しか存在しません．そのため，画像データの保存量はフルカラーRGBの1/3で済むことになります．

通常デジタル・カメラでは，フルカラーRGB画像データを，例えばJPEGで何分の1かに圧縮しています．しかし，Bayerパターン・カラー・フィルタ配列 のグレイスケール画像データであれば，圧縮なしでもそれなりにファイル容量の削減効果があります．Bayerパターン・カラー・フィルタ配列のグレイスケール画像データは，「RAW(生)データ」[注3]と呼ばれています．RAWデータそのものを画像圧縮すると，さらに効率的です．これについては，既に原理的な提案がされているので後で詳しく解説します．

注1：詳細はFoveonのWebサイト(http://www.foveon.com/)やシグマのWebサイト(http://www.sigma-photo.co.jp/)を参照．
注2：FinePix S5200は，2006年9月に出荷を終了している．
注3：本稿では，RAWデータのほかに，「RAW画素データ」，「RAW画像データ」，「RAWモード・データ」という言葉を用いている．RAW画素データという場合は，センサの画素に重点を置いている．RAW画像データの場合は，センサから読み出されて画像データとして存在している状態を指す．RAWモード・データは，メーカが作成した固有の機能であることを強調した表現．

$$G_0 = \frac{G_2 + G_4 + G_6 + G_8}{4}$$

（a）Gの線形補間

R1	R2	R3
R8	R0	R4
R7	R6	R5

$$R_2 = \frac{R_1 + R_3}{2} \quad R_4 = \frac{R_3 + R_5}{2}$$

$$R_6 = \frac{R_5 + R_7}{2} \quad R_8 = \frac{R_1 + R_7}{2}$$

$$R_0 = \frac{R_1 + R_3 + R_5 + R_7}{4}$$

（b）R（B）の線形補間

図3　画素の線形補間
（a）にGの線形補間を示す．平均を求めることでG_0が定まる．（b）はRの線形補間であるが，Bについてもやり方はRの場合と同じ．

②デジカメの“もったいない”画像処理

Bayerパターン・カラー・フィルタ配列のイメージ・センサからは，グレイスケールの輝度をもったRAW画素データが取り出されます．このRAWデータから欠損色の画素値を補間し，フルカラーRGB画像を作り出します．この処理を「デモザイキング（demosicking）」と呼びます．欠損色の画素値の補間は，周辺画素からの平均値を求める程度の処理です．

上述したように，デジタル・カメラでは一般にデモザイキング後のフルカラーRGB画像データをJPEGなどで画像圧縮してファイル保存します．これらの処理はすべてデジタル・カメラの内部で行われ，ほとんどの場合ユーザは画像圧縮されたファイル・データしか取り出すことができません．たまに，画像圧縮なしのTIFFファイル・フォーマットで画像データを取り出せる機種がありますが，デモザイキング後の画像データであり，RAWデータではありません．ところが最近，主に一眼レフのデジタル・カメラでは，RAWデータをファイル形式で取り出せる機能（RAWモードと呼ばれる）を備えた機種が発売されるようになりました．残念ながら安価なコンパクト・サイズのデジタル・カメラではまだこの機能はサポートされていないようです．

●一般的なデモザイキングの方法

ここから，カラー・フィルタ配列画素上の欠損色を補間するデモザイキングについて簡単に解説します．筆者も調査してみたのですが，結局，デジタル・カメラ・メーカ各社がどのようにしてデモザイキングを行っているのか具体的には分かりませんでした．そこで，ここではよく知られている方法について説明します．

P1	P2	P3
P8	P0	P4
P7	P6	P5

図4
画素P_0の補間
画素P_0を補間で求める場合，エッジの方向を考慮するため，P_iに対してe_iという重みを付ける．

一般に，カラー・フィルタ配列の欠損色を補間するアルゴリズム（デモザイキング）として，次のような方法が挙げられます．

1）各色独立に補間法を適用する方法
（例えば，バイリニア，バイキュービック法など注4）

2）色の比で補間法を適用する方法
（Gに対するRまたはBの比を利用する）

3）エッジの方向を検出して必要に応じて補間法を適用する方法

図3（a）に示すように，**1）**の方法はバイリニアといっても，単に平均を求めるだけです．BもRと同じように求められます．

2）と**3）**については，これらを組み合わせたKimmelの方法(4)と呼ばれるものを例にとって説明します．この方法は，「同じ物体内の色比は一定である」という，かなりシンプルなカラー画像のモデルを利用するものです．例えば図4に示す画素P_0を補間で求める場合，エッジの方向を考慮するため，P_iに対してe_iという重みを付けます．ここでまず，密度の高いG（GはRやBの2倍）の画像だけを用いて補間を始めます．

$$G_0 = \frac{e_2G_2 + e_4G_4 + e_6G_6 + e_8G_8}{e_2 + e_4 + e_6 + e_8} \quad \cdots\cdots(1)$$

式（1）によりGの補間がすべて終わったら，Gに対する色比を用いてRとBを以下のように求めます．

$$R_0 = G_0 \frac{e_1\dfrac{R_1}{G_1} + e_3\dfrac{R_3}{G_3} + e_5\dfrac{R_5}{G_5} + e_7\dfrac{R_7}{G_7}}{e_1 + e_3 + e_5 + e_7} \quad \cdots\cdots(2)$$

注4：バイリニア，バイキュービック法など，各種アルゴリズムの詳細については，参考文献（7）～（9）を参照のこと．

$$B_0 = G_0 \frac{e_1 \frac{B_1}{G_1} + e_3 \frac{B_3}{G_3} + e_5 \frac{B_5}{G_5} + e_7 \frac{B_7}{G_7}}{e_1 + e_3 + e_5 + e_7} \quad \cdots\cdots\cdots\cdots\cdots\cdots (3)$$

補間は，変化の少ない方向の画素に対して行います．そこで，Gに対して水平方向に$H = |G_8 - G_4|$，垂直方向に$V = |G_2 - G_6|$のように差を求めます．そして重みe_iを，

$$\begin{cases} G_0 = \dfrac{G_2 + G_6}{2} & (H > T > V) \\ G_0 = \dfrac{G_4 + G_8}{2} & (V > T > H) \\ G_0 = \dfrac{G_2 + G_4 + G_6 + G_8}{4} & (\text{上記以外}) \end{cases} \quad \cdots\cdots\cdots\cdots (4)$$

となるように決めます．ここで，Tは判定のためのしきい値です．変化の少ない方向は連続していて，変化の大きい方向は不連続であると考えられるので，次式のようにe_iが求められます．

$$\begin{cases} e_2 = e_6 = 1,\ e_4 = e_8 = 0 & (H > T > V) \\ e_2 = e_6 = 0,\ e_4 = e_8 = 1 & (V > T > H) \\ e_2 = e_6 = e_4 = e_8 = 1 & (\text{上記以外}) \end{cases} \quad \cdots\cdots\cdots\cdots\cdots (5)$$

なお，RとBについては重みを次のように定めます．

$$\begin{cases} e_1 = e_3 = e_5 = e_7 & (P_5\text{の位置の補間}) \\ e_2 = e_6 = 1,\ e_4 = e_8 = 0 & (P_2\text{と}P_6\text{の位置の補間}) \\ e_2 = e_6 = 0,\ e_4 = e_8 = 1 & (P_4\text{と}P_8\text{の位置の補間}) \end{cases} \quad \cdots (6)$$

さらに，斜め方向の変化を取り入れるために，

$$\begin{cases} D_x(P_0) = \dfrac{P_8 - P_4}{2}, \quad D_y(P_0) = \dfrac{P_2 - P_6}{2} \\ D_{xd}(P_0) = \dfrac{P_3 - P_7}{2\sqrt{2}}, \quad D_{yd}(P_0) = \dfrac{P_1 - P_5}{2\sqrt{2}} \end{cases} \quad \cdots\cdots\cdots\cdots\cdots (7)$$

を求めます．G_0を求めるともう少し正確になるので，次式に示すようになります．

$$\begin{cases} D_{xd}(P_0) = \max\left\{ \dfrac{|P_3 - P_0|}{\sqrt{2}}, \dfrac{|P_7 - P_0|}{\sqrt{2}} \right\}, \\ D_{yd}(P_0) = \max\left\{ \dfrac{|P_1 - P_0|}{\sqrt{2}}, \dfrac{|P_5 - P_0|}{\sqrt{2}} \right\} \end{cases} \quad \cdots\cdots\cdots\cdots\cdots\cdots (8)$$

そして重み関数e_iは，

$$e_i = \frac{1}{\sqrt{1 + D^2(P_0) + D^2(P_i)}} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (9)$$

で求められます．Kimmel氏（Kimmelの方法の発案者）は，こうして求めた値をさらに改良しようとしていますが，本稿ではデモザイキングの概要を理解していただくのが目的なので詳細は割愛します．詳細は参考文献(4)を参照してください．

● デモザイキングにキュービック法を適用

以上，従来のデモザイキングの方法を説明しましたが，ここで演算器設計の立場から見ると，ある問題点を指摘できます．

問題点とは，色比を求めるための除算（式(1)～式(3)），重みを求めるための開平と除算（式(7)～式(9)）などです．例えばCPUの除算命令や開平命令を使うと，演算ビット長に比例して処理に必要なサイクル数も長くなります．専用演算器で実現すればよいのですが，回路規模が増加します．RAWモードは16ビット長で扱うので，テーブル化も無理があります．

ということで，筆者は新しい視点で考えることにしました．画像の拡大・縮小スケーリングに用いられる補間法の一つにバイキュービック・スプライン法があります(7)～(9)．バイキュービック・スプライン法を利用すれば，画像をシャープに，（たとえ斜め方向でも）きれいに拡大できます．そこで，デモザイキング法の**1)**に対して，バイキュービック・スプライン補間法を試してみました．

ところで，デジタル・カメラが出力する，デモザイキング後のフルカラーRGBの画像データに，バイキュービック・スプライン法を適用してもあまり効果はでません．JPEGで圧縮している場合，画質が劣化しているからです．また，RAWモードで撮影し，デモザイキングして圧縮なしで画像データ[注5]を取り出しても効果は期待できません．これらはすべて，上記で説明したようにデモザイキング処理の方法が，基本的には1次（画素間の平均の場合）または2次（画素間の差分をとっている場合）の補間であることから，それらのデータをもとにして3次補間を行っても意味のないことだからです．

そこで，**図5**に示すように，筆者は従来のようなデモザイキング処理を行わず，Bayerパターン・カラー・フィルタ配列のRAWデータ（グレイスケール）から直接，バイ

注5：デジタル・カメラ・メーカが提供するツール，またはPhotoshop CS2などを利用して，TIFFまたはビットマップ・フォーマットで保存したもの．

キュービック・スプライン法を適用して画像の拡大処理を行うことにしました．

ところが，次に説明するように，画像の拡大処理以前にRAWデータを入手するために悪戦苦闘することになりました．そしてこの過程で分かったことは，現状のデジタル・カメラが，せっかく高精細に撮影した画像の多くの情報を捨て去り，ユーザに提供しているということです．なんとも「もったいない」ことをしているわけです．

図5　画像の取得手順

(a)は，カラー画像を拡大するときに，従来のデモザイキング処理後のカラー画像を拡大処理する場合．(b)は，筆者が考えたカラー・フィルタ配列画像から直接拡大処理する方法．

● RAWデータが取り出せない!？

RAWモードをサポートしている機種を用いてユーザがRAWデータを扱うためには，メーカが用意している加工ソフトウェア（ビューワ）を使います．また，米国Adobe Systems社のPhotoshop CS2などの画像編集用アプリケーション・ソフトウェアを利用することも考えられます．これらソフトウェアは，RAWモード・ファイル・データを適当に加工し，デモザイキングして，最終的なフルカラーRGB画像に仕上げるのが目的です．そのため，Bayerパターン・カラー・フィルタ配列のRAWデータをそのまま出力して，ユーザがじかに加工処理したいときに必要なサポートがありません．そこで，ユーザみずからがRAWモードの画像ファイルのフォーマットを調べて，データを取り出す必要があります．

しかし，RAWデータを格納するファイル・フォーマットは，デジタル・カメラ・メーカごとに定義されていて（場合によっては同じメーカの機種間でも異なることがある），さらに未公開であるため扱いが容易ではありません．例えば，ニコンはデータに暗号をかけていますし，キヤノンはデータに圧縮をかけています．富士フイルムはフォーマットが比較的に単純で解析しやすいのですが，独特のカラー・フィルタ配列（ハニカム構造）なので，デモザイキングが通常と異なります．そのような訳で，ユーザがRAWデータを直接，かつ容易に扱えるような適当な機種は見当たりません．

③ RAWデータを取り出してみる

そこで，Dave Coffin氏が作製したフリー・ソフトウェア「dcraw」[注6]を利用するという手があります．本ソフトウェアは，各デジタル・カメラ・メーカの多機種にわたって対応しています．最新機種の場合，メーカが提供するソフトウェア以外は未対応ということになりがちですが，dcrawはCoffin氏の個人的な解析によりいち早く対応しているようです．

このソフトウェアを使ってBayerパターン・カラー・フィルタ配列のRAWデータをそのまま出力するには，少々テクニックが必要です．ここでは，キヤノンのデジタル・カメラを例にとってRAWデータの出力方法を説明します．また，ユーザ自らがファイル・フォーマットを解読してみるという手段もあります．比較的フォーマットが解読しやすかった富士フイルムのデジタル・カメラを例に，そのときの解読のノウハウをお教えします．

● dcrawで実際にデジカメのRAWデータを取り出す

キヤノンのデジタル・カメラ（EOS Kiss デジタルX）の場合，RAWデータは可逆圧縮（ロスレス圧縮）されており，拡張子.CR2のRAWモード・ファイルに格納されています．圧縮ファイルを解読して，デコードするにはかなり手間がかかりそうなので，あきらめてdcrawを用いることにしました．dcrawは，対応機種のRAWモード・ファイルを入力すると，PPM（Portable Pixel Map）形式またはPGM（Portable Gray Map）形式のファイルを出力します[注7]．ここでは，グレイスケールのBayerパターン・カラー・フィルタ配列のRAWデータで出力することが目的なので，PGM形式を利用します．PGM形式のファイル・フォーマットの出力の場合，ファイル・ヘッダが，

```
［ファイル形式識別記号］\n
#［コメント］\n
［width］［height］\n
```

注6：Dave Coffin氏のWebサイト（http://www.cybercom.net/~dcoffin/dcraw/）からダウンロードできる．

注7：PPMやPGMはUNIXにおいて用いられるビットマップ画像フォーマット．PPMはカラー，PGMはモノクロ（グレイ・スケール）．

図6　dcrawで16ビットのグレイスケールPGM形式のファイルに変換した例

キヤノンのデジタル・カメラ（EOS Kiss デジタルX）のRAWモード・ファイルをdcrawに入力し，PGM形式で取り出した．ここで注意が必要なのは，取り出したデータは下位けた，上位けたの順番で格納するリトル・エンディアン形式であるということ．例えば，データの並びが“0F8D0C”の場合，開始アドレスが奇数アドレス（000013）から始まっていれば，下位けた，上位けたの順番になっているので，データ値は“0F8D”となる．もし偶数アドレス（000014）から始まっているとすると，下位けた，上位けたの順に並べたときデータ値は“8D0F”となる．“0C”は次の16ビット単位のデータなので，データ値を“8D0C”と解釈してはならない．

```
［最大輝度値］\n
［画像データ部］
```

となっています（図6）．\n（16進コード0A）は改行を示しています．ファイルの拡張子がPGM形式なので，ファイル形式識別記号はP5（グレイスケールRAW形式）で示されています．この後に必要なら#でコメントが入りますが，この図の場合はありません．画像サイズとして，width（幅）3906画素，height（高さ）2602画素を表しています．最大輝度値が65535なので，改行コードの後にすぐ画像データ部がバイナリ形式の16ビット単位（16進数表記）で入っています．

ここで気を付けなければいけないのは，画像データ部の開始アドレスです．図6の場合，画像データ部は奇数アドレス（000013）から始まっているので，一つの画像データ（16進数，16ビット）は“0F8D”となります．これがもし偶数アドレス，例えば000014から始まっていれば，画像データは“8D0F”のように順番は逆になります．Windows OSのパソコンと同じように，リトル・エンディアンという格納形式になっています．

さて，dcrawを実際に使用するには適当なCコンパイラが必要ですが，これは例えばCygwinなどが容易に入手できます．CygwinのWebサイト（http://cygwin.com/）からダウンロードして，GCCを用いれば十分です．コンパイルのコマンドは，

```
$ gcc dcraw.c /lib/liblcms.a /lib/libjpeg.a
```

です．デフォルトでa.exeという実行ファイルが作成されます．実行ファイル名を，例えばdcraw.exeにしたければ，

```
$ gcc -o dcraw dcraw.c /lib/liblcms.a /lib/libjpeg.a
```

とします．ここで，ライブラリliblcms.aとlibjpeg.aが必要なので，Cygwinのインストール時に忘れずにダウンロードしておいてください．もっとも，libjpeg.aはJPEG用のライブラリなので，今回のようにRAWデータのファイルを扱うときには直接関係がないのですが，コンパイルには必要です．dcrawの中でJPEGを扱うこともあるからです．liblcms.aは，Little CMS（color management system）[注8]です．

さて，dcrawの実行ファイル（.exe）のコマンドですが，RAWデータのファイル名がfilename.CR2とすると，

```
$ ./a -d -4 filename.CR2（デフォルトa.exeの場合）
```

または，

```
$ ./dcraw -d -4 filename.CR2（dcraw.exeの場合）
```

となります．オプション・コマンドの意味は，

- -d：ドキュメント・モード
（グレイスケール，かつ補間なし）
- -4：16ビット出力のガンマ補正付き
（指定なしのとき8ビット出力）

です．キヤノンのカタログ仕様では，RAWモード・データは12ビットですが，dcrawで変換するとガンマ補正が付いて，その段階で16ビットで出力されます．

注8：Little CMSとは，印刷などにおけるカラー・マネージメント処理を，コンパクトに最適化したライブラリ．オープン・ソースで配布されているもの．詳細は，「http://www.littlecms.com/」を参照．

```
           +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F   0123456789ABCDEF
0000:0000  46 55 4A 49 46 49 4C 4D-43 43 44 2D 52 41 57 20   FUJIFILMCCD-RAW
0000:0010  30 32 30 31 46 46 33 39-33 30 30 31 46 69 6E 65   0201FF393001Fine
0000:0020  50 69 78 20 53 35 32 30-30 20 20 00 00 00 00 00   Pix S5200  .....
0000:0030  00 00 00 00 00 00 00 00-00 00 00 00 30 31 30 30   ............0100
0000:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0000:0050  00 00 00 00 00 00 00 94-00 09 B4 7B 00 09 B5 12   ...........{....
0000:0060  00 00 0F 0E 00 09 C4 20-00 A1 4D 00 00 00 00 00   ....... ..M.....
0000:0070  00 A1 4D 00 00 00 00 00-00 00 00 00 00 00 00 00   ..M.............
0000:0080  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0000:0090  00 00 00 00 FF D8 FF E1-26 DA 45 78 69 66 00 00   ........&.Exif..

0009:B510  00 00 00 00 00 59 01 00-00 04 0A D8 07 70 01 10   .....Y.......p..

0009:C400  00 80 00 80 00 80 00 80-AF 02 00 04 00 00 00 40   ...............@
0009:C410  AF 03 00 04 00 40 00 40-AF 04 00 04 00 80 00 40   .....@.@.......@
0009:C420  00 20 00 20 00 20 00 20-00 20 00 20 00 20 00 20   . . . . . . . .
0009:C430  00 20 00 20 00 20 00 20-00 20 00 20 00 20 00 20   . . . . . . . .
0009:C440  00 20 00 20 00 20 00 20-00 20 00 20 00 20 00 20   . . . . . . . .
0009:C450  00 20 00 20 00 20 00 20-00 20 00 20 09 05 E7 06   . . . . . .....
0009:C460  18 05 B2 07 7B 05 D8 06-6C 05 00 07 BF 05 6A 07   ....{...l.....j.
0009:C470  33 05 45 07 A4 05 70 07-84 05 46 07 A7 05 0A 07   3.E...p...F.....
0009:C480  39 05 B3 07 CC 05 D2 07-A5 05 CE 07 58 05 EB 07   9...........X...
0009:C490  64 05 DA 06 34 05 11 07-34 05 22 07 2D 05 43 07   d...4...4.".-.C.
0009:C4A0  CE 05 5C 07 75 05 70 07-9B 05 D9 07 22 05 41 07   ..¥.u.p....."A.
0009:C4B0  32 05 84 07 C0 05 77 07-59 05 4A 07 C0 05 24 07   2.....w.Y.J...$.
0009:C4C0  6D 05 2E 07 9B 05 89 07-DF 05 36 07 30 05 3C 07   m.........6.0.<.
0009:C4D0  9F 05 5C 07 D4 05 44 07-21 05 38 07 7D 05 8A 07   ..¥...D.!.8.}...
```


図7　ファイル・フォーマットの解読
富士フイルムのデジタル・カメラ（FinePix S5200）のRAWモード・ファイル・フォーマットをバイナリ・エディタで出力させた様子．

● 自分でフォーマットを読み解く

次に，富士フイルムのデジタル・カメラ（FinePix S5200）を例に挙げて，RAWデータのファイル・フォーマットの解読方法を説明します．

本デジタル・カメラのRAWモード・ファイルの拡張子は.RAFです．図7に，バイナリ・エディタで表示した様子を示します．ヘッダ情報の後のアドレス0x0064にデータ“0x0009C420”があります．これはRAWデータが格納されているアドレスへのポインタ（ファイル先頭からのオフセット値）なので，RAWモード・ファイルごとにここを見て，RAWデータの格納位置を知ります．

また，アドレス0x005Cのデータ“0x0009B512”は，配列データの幅（0x0770）と高さ（0x0AD8）が入っている場所へのポインタとなっています．ここで注意しなければならないのは，横幅の長い画像だったにもかかわらず幅の方が小さい値が入っていることです．この理由は後述のハニカム構造の説明の中で明らかになりますが，最初はハニカム構造のRAW画素データをどのように，どの順番で格納しているのか分かりませんでした．

そこで，まず背景が純白の写真を撮り，これをもとに調べてみたところ，データが16ビット単位で“0xFF3F”を繰り返して格納されていることが分かりました．富士フイルムのカタログ仕様ではRAWモードのビット長は公開されていませんでしたが，これから14ビット（0x3FFF＝0011 1111 1111 1111；実際のデータ値は0xFF3Fの順序を逆にしたものになることに注意）であることが分かりました．さらに，上位15，16ビットには“00”が埋め込まれていて使われていません．

純白は露出をオーバさせれば簡単に撮れますが，純粋な赤，緑，青を撮るのは難しいことです．そこで，逆にバイナリ・エディタを利用してデータを書き込み，色の出具合をチェックしてデータの並び具合を推測してみました．その結果，1行目にBとR，2行目にGとG，3行目にRとB，4行目にGとGと並んでおり，それ以降はこのパターンを

繰り返していることが分かりました(詳細は後述).

④ RAWデータを直接用いた画像拡大スケーリング

以上説明したように，最近のデジタル・カメラでは，RAW画像データ値を表現するビット長は，例えば12ビットや14ビットと，8ビットを超えています．RGBをそれぞれ8ビットで表現している通常のディスプレイに表示するには，8ビット長に正規化する必要があります．これらは，画質調整と絡んでおり，工夫の余地があります．ここで筆者が強調したいのは，画像処理については，デモザイキング後のフルカラーRGBの8ビットを用いるのではなく，多くの元の情報量(例えば，8ビットを超える精度)を保持しているRAWデータを直接用いるべきであるということです．

● Gの補間では45°の回転が必要となる

上述したように，Bayerパターン・カラー・フィルタ配列によって取り出されたグレイスケール画素値は，その後，欠損色の周辺画素の平均をとる程度で補間(デモザイキング処理)され，フルカラーRGB画像として出力されます．そのため，このフルカラーRGB画像を用いた拡大処理では，鮮明な大きい画像は望めません．

鮮明な大きい拡大像を得るためには，Bayerパターン・カラー・フィルタ配列のグレイスケール画素値を直接用いて，任意位置を補間する方法が有効であると考えられます．ただし，この方法は次のような理由から処理が複雑になります．

- RGB各色で参照する画素が異なること
- 45°回転した配置で参照する画素があること
- 補間比が3色すべてで異なること

通常のデモザイキングのように，周辺画素の平均をとる程度の補間処理では，参照する画素の選択はそれほど難しいものではありませんでした．つまり，本稿で紹介するグレイスケール画素値を直接利用する方法の処理の様相は，通常のデモザイキングとは全く違うことに注意してください．

RとB(ハニカム構造の場合はG)の画素については，**図2**に示したように水平・垂直方向に配置されているので，従来の補間法が適用できます．しかし，G(ハニカム構造の場合はB，R)の画素については，どのように補間すればよいか少し考える必要があります．例えば，**図8**において補間点をXとすると，平行四辺形ABDEの頂点にある画素を参照するのか，それとも45°回転した正方形ABCDの頂点にある画素を参照するのがよいのか迷ってしまいます．また，たとえ平行四辺形ABDEを参照画素とするとしても，水平方向の補間について補間点に(r, s)対を選ぶのか，それとも(p, q)対を選んで斜交座標系で行うのがよいのかで悩むことになります．

そこで，補間点に近い位置にある画素ほど補間点に近い情報を持っていることから，補間の参照画素としては，補間点に最も近い画素から4個選べばよいことになります．厳密に参照画素点の勢力範囲を求めると，**図8**に示すように三角の領域DFG内は，BよりもEの方が強いといえます

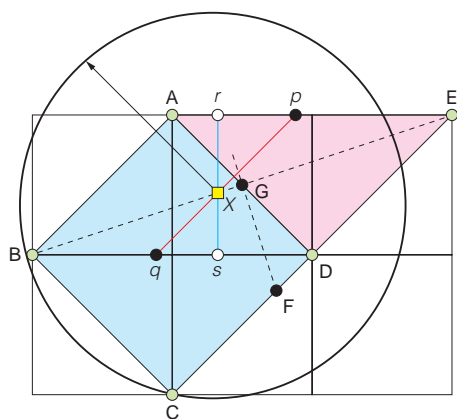



図8　参照点の選び方と補間方法
G(ハニカム構造の場合はB，R)の画素の補間は難しい．例えば，図において補間点をX(黄色い四角の部分)とすると，平行四辺形ABDEの頂点にある画素を参照するのか，それとも45°回転した正方形ABCDの頂点にある画素を参照するのがよいのか迷ってしまう．

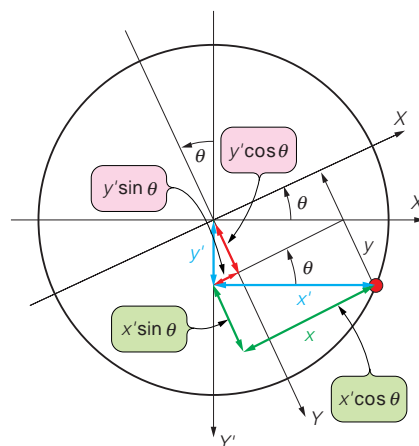



図9　回転式の導出
G(ハニカム構造の場合はB，R)の画素については，補間するために45°の回転操作が必要となる．ここでは，π/4＝45°の回転式ではなく，任意角θの回転式を導くようにする．

が，そのようなことを考慮した補間法や制御は複雑になります．そこでもっとシンプルに考えて，正方形ABCDの頂点の参照画素を選びます．補間点xにより近いため，平行四辺形ABDEの頂点の参照画素よりも適していることが分かります．従って，G（ハニカム構造の場合はB，R）の画素については，補間するためには45°の回転操作が必要です．$\pi/4=45$°の回転式だけなら簡単なのですが，ここではついでに図9に示すように任意角θの回転式を導いておきます．すなわち，以下の式を求めることになります．

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x'\cos\theta - y'\sin\theta \\ x'\sin\theta + y'\cos\theta \end{pmatrix} \quad \cdots\cdots(10)$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

$$= \frac{\sqrt{2}}{2} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{\sqrt{2}}{2} \begin{pmatrix} x'-y' \\ x'+y' \end{pmatrix} \quad \cdots\cdots\cdots\cdots\cdots\cdots(11)$$

● 参照画素の整数座標値の求め方

では，実際にどのようにして各RGBを補間するのかについて説明します．

図10はあるBayerパターン・カラー・フィルタ配列の画素配置における任意位置での補間の様子を示しています．配列画素の整数座標値を(i, j)によって表すと，各RGBについて，$R=(2i, 2j)$，$B=(2i+1, 2j+1)$，$G=(2i+1, 2j)$または$(2i, 2j+1)$となります．説明を簡単にするため，拡大の場合の原点はB(1, 1)にあるとします．そして，図10に示す範囲の位置に実数座標値の補間点(x, y)があるとします．補間のために参照すべき画素は，各RGBで異なります．バイリニアの場合の4個の参照画素の座標は次のようになります．

- R：(2, 2)，(2, 4)，(4, 2)，(4, 4)
- B：(1, 1)，(1, 3)，(3, 1)，(3, 3)
- G：(3, 2)，(2, 3)，(3, 4)，(4, 3)

次に，任意の補間点(x, y)が与えられたときの，各RGBの参照画素の整数座標値(i, j)の求め方について検討してみます．

補間点(x, y)の小数部を切り捨てて整数化すればよいのですが，Rは偶数$(2i, 2j)$，Bは奇数$(2i+1, 2j+1)$の座標位置になっているので注意が必要です．すなわち，Rについては$(x/2, y/2)$としてから整数化しますが，Bについては$((x-1)/2, (y-1)/2)$としてから整数化します．残るGについては45°回転する必要があるので非常に厄介です．以下に詳しく説明します．

まず，R（またはB）について，図11に示すように4個の補間参照点の左上の位置にある点(i, j)を，R（またはB）

3

図10　Bayerパターン・カラー・フィルタ配列画素から直接補間

バイリニアの場合，4個の参照画素の座標は，R：(2，2)，(2，4)，(4，2)，(4，4)，B：(1，1)，(1，3)，(3，1)，(3，3)，G：(3，2)，(2，3)，(3，4)，(4，3)となる．

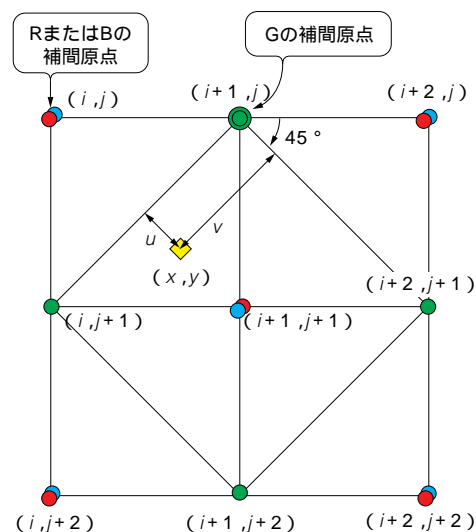



図11　Gは45°回転して補間

45°回転するということは，(x, y)座標から(u, v)座標に変換される．補間範囲の原点位置$(i+1, j)$を求める．長さは$\sqrt{2}$倍されるが，補間の割合は0と1の間にあるので，正規化$(1/\sqrt{2})$によって，$\sqrt{2}$が消える．

の補間原点と呼ぶことにします．Gについては，上下左右4個の補間参照点の中で，上の位置にある点を補間原点(i＋1，j)とします．

図9から(x，y)→(u，v)への回転は反対方向(−45°)の回転であることに注意し，式(2)から点(i+1，j)を原点とする回転式を求めると，次式のようになります．

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{\sqrt{2}}{2}\begin{pmatrix} (x-i-1)+(y-j) \\ -(x-i-1)+(y-j) \end{pmatrix} \quad \cdots\cdots(12)$$

G間の距離は$\sqrt{2}$ですが，補間は0と1の間に正規化された区間[0，1]で行うことから，式(12)を$\sqrt{2}$で割って，式(13)のように正規化します．

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \frac{1}{2}\begin{pmatrix} (x-i-1)+(y-j) \\ -(x-i-1)+(y-j) \end{pmatrix} \quad \cdots\cdots(13)$$

これで係数$\sqrt{2}$は消去されます．このことから，

$$\frac{1}{2}\begin{pmatrix} x+y-1 \\ y-x+1 \end{pmatrix}$$

⇓ 整数化

$$\begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} \left\lfloor \frac{1}{2}(x+y-1) \right\rfloor \\ \left\lfloor \frac{1}{2}(y-x+1) \right\rfloor \end{pmatrix} = \frac{1}{2}\begin{pmatrix} i+j \\ j-i \end{pmatrix} \quad \cdots\cdots(14)$$

のように，整数化(m，n)を行えば，

$$\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} m-n \\ m+n \end{pmatrix} \quad \cdots\cdots(15)$$

によって，(i，j)が求まり，Gの補間原点(i＋1，j)が得られます．

ここで，注意すべきことは，式(14)において負の値をとる場合があるということです．図12に示すように，実数の整数化において，正の範囲では単純に小数部の切り捨てを行えばよいのですが，負の範囲も正の場合と同じ方向に小数部の切り捨てを行う必要があります．

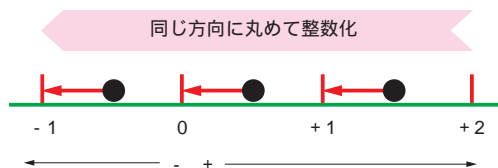



図12　整数化のために丸める方向
実数の整数化において，正の範囲では単純に小数部の切り捨てを行えばよいが，負の範囲も正の場合と同じ方向に小数部の切り捨てを行う必要がある．

数学的には，実数xに対して，xを超えない最大の整数iを求めます．ガウスの記号$\lfloor\ \rfloor$を用いて，$i=\lfloor x \rfloor$と表されます($\lfloor\ \rfloor$はfloor関数とも呼ばれる)．例えば−1＝$\lfloor -0.10101 \rfloor$の場合，固定小数点数−0.10101を2の補数で表現すると，11.01011(最上位けた符号部＋整数部1けた＋小数部5けた)となります．小数部を単純に切り捨てると11.00000となり，−1＝11.00000と表せます．

符号付き固定小数点数A(整数部mけた，小数部nけた)は次式で表現されます．

$$A = -a_m 2^{m-1} + \sum_{i=0}^{m-1} a_i \cdot 2^i + \sum_{j=1}^{n} a_{-j} \cdot 2^{-j} \quad \cdots\cdots(16)$$

これをガウスの記号を用いて表現すると，

$$\lfloor A \rfloor = -a_m 2^{m-1} + \sum_{i=0}^{m-1} a_i \cdot 2^i \quad \cdots\cdots(17)$$

になることから，2の補数で固定小数点表現されていれば(符号付き)，小数部の単純な切り捨てで大丈夫なことがわかります．

ただし，絶対値による小数部の単純な切り捨ては行えないので注意が必要です．このような問題が起こるのは，数を浮動小数点で表現しているときです．例えば，C言語では浮動小数点の整数化にint関数を使いますが，そのときにはこの問題が生じます．浮動小数点表現は，数を絶対値表現して符号ビットで正負の区別を表現しています．小数部の丸め方については，プログラミングの処理系やライブラリによって違いがある場合もあるので，気を付けてください．

以上のような考えのもとで，各RGBの補間を制御し，求めた補間画像を1枚に重ね合わせます．

● ハニカム構造では画素を配置していない格子点がある

富士フイルムが独自に採用しているハニカム構造のカラー・イメージ・センサは，画素の座標配置の観点から見れば図13に示すようになります．今まで説明してきたものをちょうど45°回転すれば，ハニカム構造の場合のBayerパターン・カラー・フィルタ配列の配置が得られます．従って，45°回転する色はGではなく，RとBになります．

ここで，少し注意する点があります．画素が格子点(整数座標)上に配置されているとすると，非ハニカム構造の

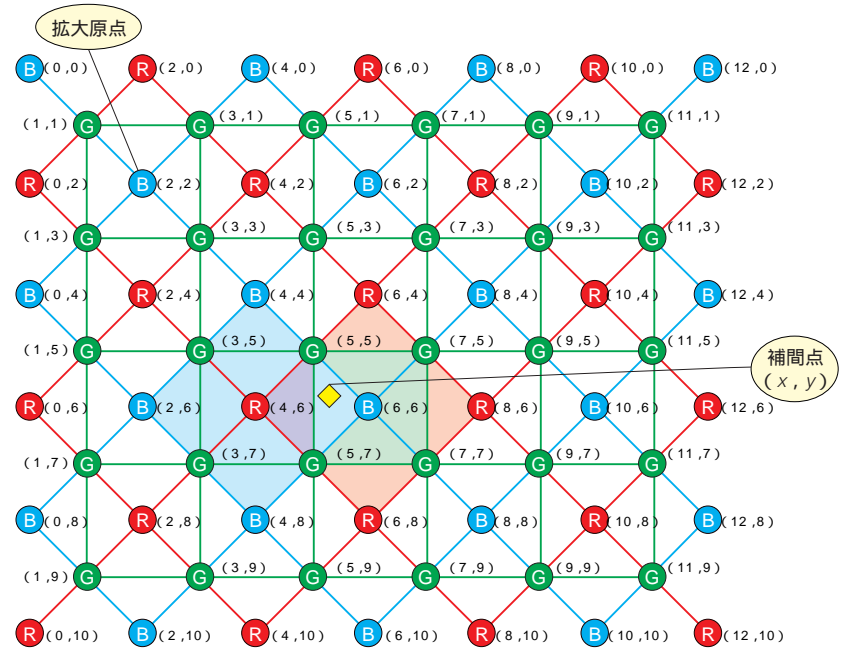



図13
ハニカム構造のBayerパターン・カラー・フィルタ配列
非ハニカム構造の場合は素子内のすべての格子点上に画素が配置されているが，ハニカム構造の場合は，図のように画素が全くのっていない格子点の場所がある．

場合は素子内のすべての格子点上に画素が配置されていました．一方，ハニカム構造の場合は，**図13**を見ると分かるように画素が全くのっていない格子点の場所があります．別の見方をすれば，画素は格子点上の一つ置きに配置されています．

このような画素が完全に欠損した部分も，補間によって埋め合わせると，非ハニカム構造の2×2倍の密度の画素データが生成されます．これがハニカム構造の画質が2×2倍良くなったかのような錯覚を引き起こしています．欠損画素を補間によって埋め合わせても，情報量が増加するわけではないので，少なくとも数学的には非ハニカム構造とハニカム構造の違いはありません．物理的には，画素間のすきまがどうなっているかですが，結局は面積当たりの実画素密度の違いに帰着することに注意してください．

以下にハニカム構造の場合の補間の扱いについて説明します（**図14**）．まず式(12)から，$(i+1,\ j-1)$を中心とする45 °の回転式は，次式のようになります．

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{\sqrt{2}}{2}\begin{pmatrix} (x-i-1)+(y-j+1) \\ -(x-i-1)+(y-j+1) \end{pmatrix} \quad \cdots\cdots(18)$$

それを正規化することによって，式(19)が得られます．

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \frac{1}{2}\begin{pmatrix} (x-i)+(y-j) \\ -(x-i)+(y-j)+2 \end{pmatrix} \quad \cdots\cdots(19)$$

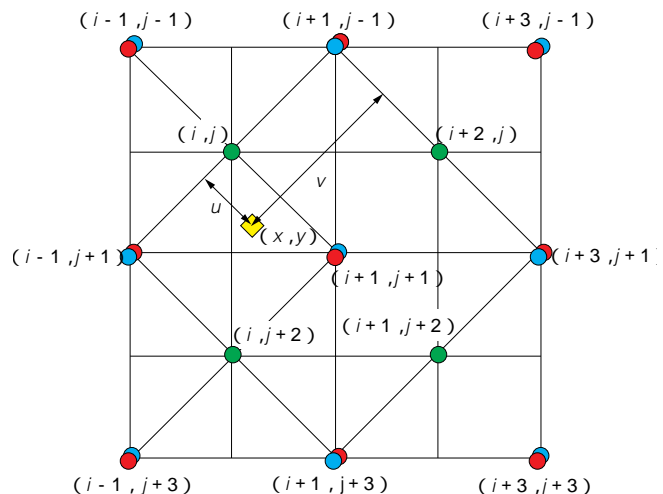



図14　ハニカム構造の場合の格子点への色配置の様子
通常の場合を45 °回転すればよいことに注意．すべての色が欠損している格子点の場所があることにも注意．Gに対して，BまたはRのどちらかが周辺に配置されている．

格子上の一つ置きに画素が欠損しているため，座標のスケールは非ハニカム構造の場合の2倍になっています．そのため，整数化のスケールは非ハニカム構造の場合は1/2ですが，ハニカム構造の場合は1/4になります．従って，実際の正規化は式(20)のようになります．

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \frac{1}{4}\begin{pmatrix} (x-i)+(y-j) \\ -(x-i)+(y-j)+2 \end{pmatrix} \quad \cdots\cdots(20)$$

そして，回転の中心$(i+1,\ j-1)$がRならば，このと

きのもう一色Bは，座標を(2，0)だけシフトすることで求められます．

5 実際のカメラ・データを用いた比較・評価

では，実際にデジタル・カメラのRAWモード画素データを用いて直接，補間拡大スケーリング表示する方法と，デモザイキングしたフルカラーRGB画像から補間拡大スケーリング表示する方法を比較します．

● 実際の開発時もまずはゾーンプレートを使う

まずは原理的な違いを見るためにゾーンプレート(人工的に描画した画像)を使って比較してみます．**図15**はキヤノン(EOS Kissデジタル X)の，**図16**は富士フイルム(FinePix S5200)の機種を想定して，オリジナルのグレイスケール・ゾーンプレートを描き，Bayerパターン・カラー・フィルタ配列の配色パターン位置に応じて各RGBの色付けを行ったものです．

図15は全体的に緑っぽくなっていますが，これはGの数がRやBの2倍あるからです．また，**図16**のハニカム構造の画像が暗くなっているのは，全画素(格子点)の半分で画素そのものが欠損しているため，画素欠損部を黒で埋めたからです．

図15(b)や**図16(b)**の最近傍法のパターンを見てもらうと，45 °回転して補間を適用している様子がよく分かります．ゾーンプレートの場合，Bayerパターンを用いないでフルカラーRGBで拡大補間表示すると，各RGBの輝度値が一致しているため厳密なグレイスケール表示になります．それに比べて，Bayerパターンを用いる場合のバイリニア法(**図15(c)**，**図16(c)**)では，RGBごとに参照画素および補間点への距離の比が異なるため，補間で求めたそれぞれの輝度の値が一致しないことからずれが生じ，それが色ずれとして現れています．

各RGBで参照画素が異なったり，補間点への距離比が異なっていても，バイキュービック・スプライン法を採用すれば補間精度が良いため，補間で求めた各RGBの輝度値がほぼ一致し，グレイスケール表示となり，色ずれが目立ちません(**図15(d)**，**図16(d)**)．

ハニカム構造の場合，格子点上で完全に欠損している画素分を補間すると，それだけで2×2倍の画像が出来上がるので注意が必要です．1×1倍の画像は2×2倍の画像を間引く(縮小する)ことで得られます．

図17は，整数化のための丸め(切り捨て)を正の範囲と負の範囲で反対の方向(絶対値)にした場合に出る症状の一例です．バイリニアとバイキュービック・スプライン法の場合を示しています．

一般に，Bayerパターン・カラー・フィルタ配列のRAWデータから直接拡大スケーリング表示するという方法を採用してプログラムまたはシステムを開発する場合，ゾーン

図15　キヤノン製デジタル・カメラのゾーンプレートの拡大表示
EOS Kiss デジタル X を想定した，Bayerパターン・カラー・フィルタ配列を用いたゾーンプレートの拡大(8×8倍)表示．

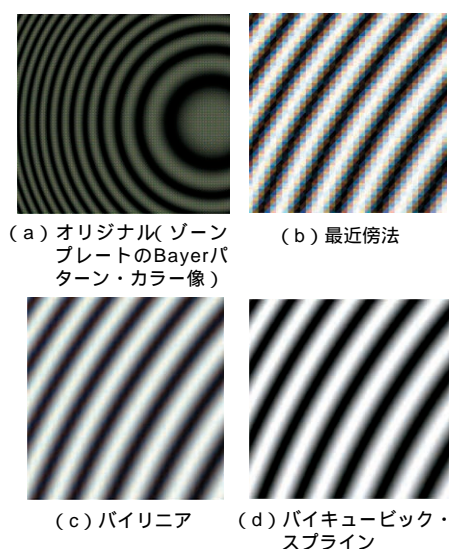
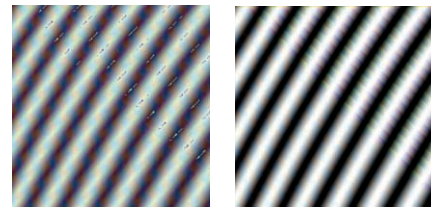



図16　ハニカム構造の場合のゾーンプレートの拡大表示
FinePix S5200を想定した，Bayerパターン・カラー・フィルタ配列を用いたゾーンプレートの拡大(4×4倍)表示．

プレートを使いながら進めるとよいでしょう．いきなり実写真のRAWモード・ファイルを使って開発すると，まともな画像がなかなか現れず，開発時間がかかってしまいがちになります．

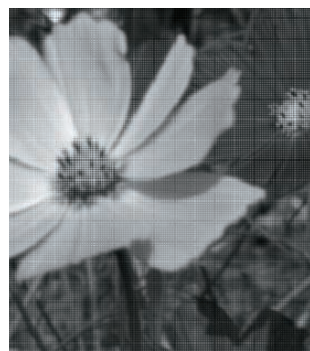
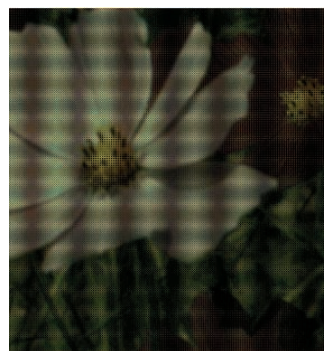
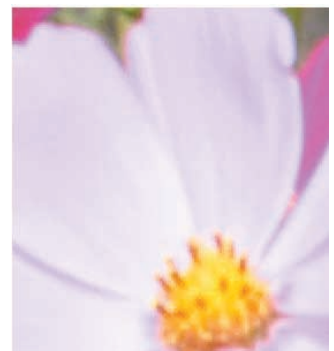
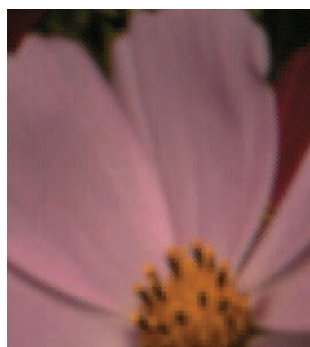
● 実際のカメラ画像で評価する

実際にデジタル・カメラのRAWモード画像データを用いて直接，拡大スケーリング表示した例を**写真1**に示します．既に説明したように，dcrawを用いて拡張子.CR2のRAWモード・ファイル・データを拡張子.pgmのPGMファイル形式に変換しています．ここで，カタログ仕様である12ビット長精度のRAW画像データは，dcraw内でガンマ補正が行われて，実質16ビット長のデータとして出力されています．

これらデータを用いて直接拡大処理したフルカラーRGB表示データは，ディスプレイには各RGB8ビット長で表示するため，16ビットから8ビットへの変換が必要です．このとき，本来なら明るさのバランスを調整して変換するのですが，**写真1(d)～(f)**では厳密にその処理を行っていません．そのため，全体的に暗い感じになっていますが，明暗差はよく出ています．これに対して，**写真1(c)**のデモザイキング処理した各8ビット長のフルカラーRGB画像データから従来ツールで拡大処理した場合は，明暗差がはっきりしません．

(a) バイリニア

(b) バイキュービック・スプライン

図17 整数化のために丸める方向を誤ったときに出る症状例
整数化のための丸め(切り捨て)を正の範囲と負の範囲で反対の方向(絶対値)にした場合に出る症状の一例．

写真2に，ハニカム構造の場合の実写真の例を示します．上述したハニカム構造独特の配置によって，2×2倍の拡大を行っても，実際はその倍の画素が表示されるので，実質は4×4倍相当になります．Bayerパターン・カラー・フィルタ配列のRAWデータにパターンの配置色を付けていますが，格子点上の画素欠損が倍あり，黒を埋め合わせ

(a) オリジナル(写真のグレイスケール像)

(b) オリジナル(写真のBayerパターン・カラー像)

(c) 従来ツールによる拡大

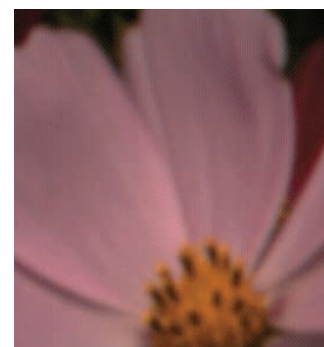
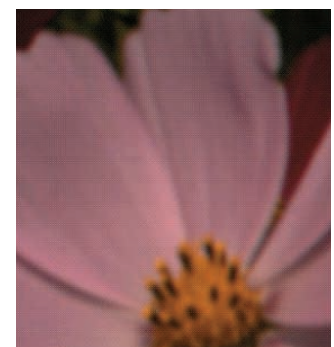
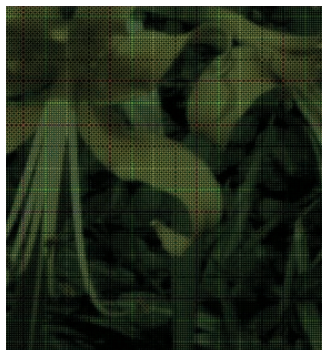
(d) 最近傍法

(e) バイリニア

(f) バイキュービック・スプライン

写真1 実写真画像の拡大表示例
キヤノンの1,000万画素のデジタル・カメラ(EOS Kiss デジタルX)で撮影した写真で，Bayerパターン・カラー・フィルタ配列のRAWデータを直接拡大スケーリング(2×2倍)させたもの．

(a) オリジナル（写真のグレイスケール像）

(b) オリジナル（写真のBayerパターン・カラー像）

(c) 従来ツールによる拡大

(d) 最近傍法

(e) バイリニア

(f) バイキュービック・スプライン

写真2
ハニカム構造を採る実写真画像の拡大表示例

富士フイルムの500万画素のディジタル・カメラ（FinePix S5200）で撮影した写真で，Bayerパターン・カラー・フィルタ配列のRAWデータを直接拡大スケーリング（2×2倍）させたもの．2×2倍の拡大を行っても，実際はその倍の画素が表示されるので，実質4×4倍相当．

たために，全体的に暗くなっています．

既に述べたように，ファイル・フォーマットを解析した結果，FinePix S5200のRAWモードの画素データのビット長は14ビットで，上位2ビットに‘0’が詰まっているので16ビット単位で格納されています．ここではそれをそのまま読み出して取り出しただけで，ガンマ補正はかかっていません．この場合のガンマ補正は別途解析して検討する必要があるためです．しかし，デモザイキング処理した各8ビット長のフルカラーRGB画像データから従来ツールで拡大処理した場合（**写真2（c）**）にはガンマ補正がかかっています．そのため，Bayerパターン・カラー・フィルタ配列のRAWデータから直接拡大した画像と見比べると，異なった印象を受けます．RAWデータから直接拡大した画像（**写真2（d）～（f）**）では，明るい部分が潰れていません．

写真3に，これまで説明してきたBayerパターン・カラー・フィルタ配列のRAWデータから，直接拡大スケーリング表示する方法を英国Celoxica社製のFPGAボード（RC300E）に実装し，拡大表示させた様子を示します．ゾーンプレートをBayerパターンで配色表示したオリジナル画像から，直接拡大スケーリング表示しています．既に作成していた従来のフルカラーRGB画像の方式を少し手直しするだけでしたが，処理が少し重くなり，いくつかの最適化を図る必要がありました．参照する画素の選択法を各RGBで変えなければならないことと，45°回転の処理が加わったからです．

写真3　FPGAボードへ実装

Bayerパターン・カラー・フィルタ配列のRAWデータから，直接拡大スケーリング表示する方法を英国Celoxica社製のFPGAボード（RC300E）に実装し，拡大表示させた様子．

⑥ RAWデータを直接用いた画像圧縮

通常，図18(a)に示すようにBayerパターン・カラー・フィルタ配列のRAW画像データをデモザイキング処理し，フルカラーRGB画像データにしてから圧縮処理を施して，ファイル保存しています．ところが，Bayerパターン・カラー・フィルタ配列のRAW画素そのものはグレイスケールであり，配列データの配置場所によって解釈してRGBの色付けを行います．そのため，データ量はフルカラーRGB画像データの1/3です．わざわざ3倍冗長なデータに拡張してから圧縮するのは無駄なことです．

では，グレイスケール(RAWデータ)そのものを単純に圧縮してよいかというと，必ずしも効率的であるとはいえません．任意の画素に対して水平・垂直に隣接する画素は異なる色のデータであるため，画素列の連続性が保たれていないからです．色ごとの配置構造の考慮が必要です．それから圧縮するにしても，

- 可逆(lossless)
- 可逆に近い(near-lossless)
- 非可逆(lossy)

といった方法が考えられます．用途によってどの圧縮法を選ぶかが大切です．

繰り返しになりますが，RAWモードの画像ファイルは通常，画素の表現ビット長が表示の8ビットよりも多い16ビットで保存されているということに注意が必要です．その意味で，正確にはRAW画像データは現状フルカラーRGB画像データの2/3のデータ量です．そのほか，撮影時の条件がいくつも格納されています．

既にRAW画素データについていくつかの圧縮方法が提案・評価されていますが，フルカラーとRAWデータの比較条件が同じビット長の画素配列になっているようなので注意が必要です．RGB各8ビットで表示することが前提ならば，非可逆圧縮でも表示で差がなければ採用できるからです．とはいっても，RAWモードは撮影時の生の情報を記録しているという意味から，可逆圧縮するのが基本です．これは一部のデジタル・カメラ・メーカ(例えばキヤノン)の機種で既に使われています．

以下に，図18(b)に示すBayerパターン・カラー・フィルタ配列のRAW画像データを，直接画像圧縮する方法について説明します．

● RAWデータを直接扱うのに適した圧縮方式を考える

RAWモード画像データの一番の特色は，画像を取り込んだときの生のデータが必要なときにいつでも使えることです．RAWデータの圧縮とは，画像の利用目的が定まったときに，利用しない不要なデータを捨て去ってもよいことから可能となる作業です．例えば，監視カメラの記録で，犯人の顔を特定するために必要な情報は絶対に捨ててはなりません．従って，後で何が必要になるのか分からないときは，記録した情報から最大限の情報を取り出せるように基本的には可逆圧縮を行います．

可逆圧縮には，ハフマン符号，算術符号，LZW(Lempel-Ziv-Welch)がよく使われます．JPEGは非可逆圧縮方式ですが，ロスレスJPEGというのがあって，これにはハフマン符号が主に使われています．算術符号は複雑であることに加えて知的所有権の問題があり，今のところ積極的に使われていません．また，JPEG-2000は画像圧縮のベースにWavelet法が使われていて，効率的に可逆圧縮できるモードがあります．

さて，これら可逆圧縮方式のどれが，Bayerパターン・カラー・フィルタ配列のRAW画像データを効率良く圧縮するのに向いているのでしょうか．最近発表されたN. Zhang氏の論文(5)がこの問いに対する一つの解決の方向を提案していると思いますので，以下に解説します．

● 2次元Wavelet変換による可逆圧縮が適するという説

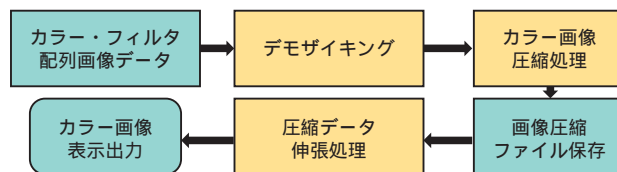
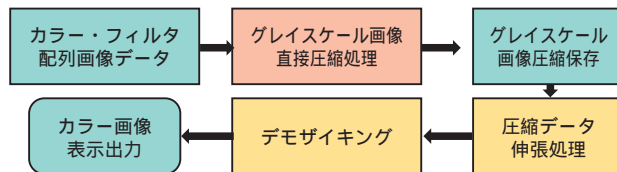
普通に思い付くのが，図10においてBayerパターン・カラー・フィルタ配列の各RGBの色別に分離して，それぞ

(a) デモザイキングして圧縮する一般的な方法

(b) デモザイキングしないで圧縮する方法

図18 カラー画像の圧縮処理

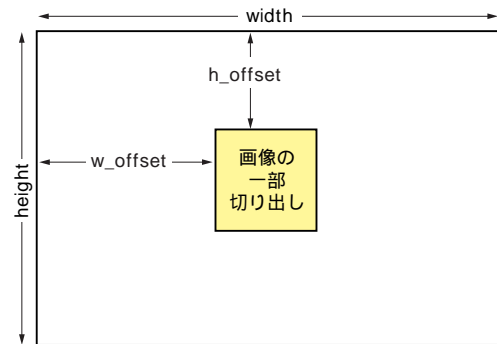
(a)は従来のデモザイキング処理後のカラー画像を圧縮処理する場合．(b)はBayerパターン・カラー・フィルタ配列のRAWデータから直接圧縮処理する場合．

COLUMN　RAWデータ読み出しのためのプログラミングのヒント

デジタル・カメラのRAWモード画像データを読み出すために，C言語でプログラミングしたコードの一部を参考として示します．

キヤノンのデジタル・カメラ（EOS Kiss デジタルX）については，dcrawを用いてPGM形式に変換したファイルを操作しています（**リストA-1**）．富士フイルムのデジタル・カメラ（FinePix S5200）については，RAWモード・ファイルからオリジナル・データを直接読み出すファイル操作を示しています（**リストA-2**）．

ここで示す例は，**図A-1**のように全画像の一部を切り出して取り出すようにしています．RGBの各色で補間のために参照する画素が異なることは本文で述べた通りです．そのため，各色の補間原点の座標を求める制御方法も参考として含めています．

なお，ここに挙げたリストは一部の要点のみなので，そっくりコピーしても動きません．コメントを参考にしながら皆さんで動くプログラムを完成してみてください．

図A-1　RAW画像データの一部切り出し

リストA-1　プログラムのための参考例1（EOS Kiss デジタルX）
ファイル操作と参照画素設定のための制御（45 °回転を含む）．

```
unsigned short bayer[hsize][wsize]; // Bayerパターン配列
                                                画像データ
COLORREF color_d[hsize][wsize];     // RGBカラー配列構造体
FILE *fp;                           // RAWモード・ファイル・
                                         データの読み出し用

int p,width,height,intensity; // p: ファイル形式識別記号,
                                         intensity: 階調数
int w_offset, h_offset;       // 切り出しオフセット
int buff_size = 24000;        // バッファ・サイズ
unsigned char *pgm;           // PGM形式ファイルを読み出し
て
                                              格納する領域

pgm=(unsigned char *)malloc(buff_size); // pgmメモリ領域
                                                   の確保

fp=fopen("filename.pgm","rb"); // ファイル・オープン:
                             filename.pgmのバイナリ読み出し
fseek (fp, 0, SEEK_SET);        // filename.pgmの先頭
                                         アドレス・セット
fscanf(fp,"P%d¥n%d %d¥n%d¥n",&p,&width,&height,&intensity);
                                // ファイル・ヘッダ部の
                                                読み出し
w_offset = 860;  // 切り出し幅オフセットのセット
h_offset = 1180; // 切り出し高さオフセットのセット
fseek (fp, 19 + h_offset*2*width + 2*w_offset, SEEK_SET);
                 // 切り出しの先頭アドレスへセット
    for (j = 0; j < hsize; j++){
         k = 0;
         fread(pgm,1,2*width,fp); // 1行分の切り出し
    for (i = 0; i < wsize; i++){
         bayer[j][i] = pgm[k] << 8 | pgm[k+1];
                                  // Bayerパターン配列画像
                                の1画素分(16ビット)取り出し
         k = k + 2;               // 次の画素へ進める
         d = bayer[j][i] >> 8;    // 表示のために上位
                                              8ビット取り出す
    // オリジナルBayer配列データの色付け
    if ( (~(i % 2)) && (~(j % 2))) color_d[j][i] =
                                  RGB(d,0,0); // red表示
    if ( (i % 2) && (~(j % 2))) color_d[j][i] =
                             RGB(0,d,0);    // green表示
    if ( (~(i % 2)) && (j % 2)) color_d[j][i] =
                             RGB(0,d,0);    // green表示
    if ( (i % 2) && (j % 2)) color_d[j][i] =
                           RGB(0,0,d);       // blue表示
    }
    }

fclose(fp); // ファイル・クローズ

    for ( j = 0; j < hsizw; j++){
    for (i = 0; i < wsize; i++){
         pDC->SetPixel(i, j, color_d[j][i]);
                          // オリジナルBayer配列カラー表示
    }
    }

delt = 1.0/zoom; // 拡大率zoomの逆数セット
    for ( j = 2; j < hzize - 2; j++){
        poy=j*delt;                  // 補間点，y座標のセット
        lr =int((poy+1.0)/2.0);      // redの補間原点:
                                                  y座標整数化
        vr = (poy + 1.0)/2.0 - lr; // redの補間比:
                                                  y座標小数部
        lb =int(poy/2.0);            // blueの補間原点:
                                                  y座標整数化
        vb = poy/2.0 - lb;           // blueの補間比:
                                                  y座標小数部
    for (i = 2; i < wsize - 2; i++){
        pox=i*delt;                  // 補間点，x座標のセット
        kr =int((pox+1.0)/2.0);      // redの補間原点:
                                                  x座標整数化
        wr = (pox + 1.0)/2.0 - kr; // redの補間比:
                                                  x座標小数部
        kb =int(pox/2.0);            // blueの補間原点:
                                                  x座標整数化
        wb = pox/2.0 - kb;           // blueの補間比:
                                                  x座標小数部
        // green部の45度回転操作
        poy=((j-i)*delt + 1.0)/2.0;
        jl = floor(poy);
             // greenの補間原点: 45 °回転y座標整数化
             // int関数でなく，floor関数を使っていることに注意
        vg = poy - jl;
             // greenの補間比: 45 °回転y座標小数部

        pox=((i+j)*delt - 1.0)/2.0;
        ik = floor(pox);
                   // greenの補間原点: 45 °回転x座標整数化
        wg =pox -ik;
                   // greenの補間比: 45 °回転x座標小数部
        // 45度回転の補間原点を求める
        kg = ik-jl+2;     // y座標
        lg = ik+jl+1;     // x座標

    }
}
```

リストA-2　プログラムのための参考2（ハニカム構造，FinePix S5200）
ファイル操作と参照画素設定のための制御（45 °回転を含む）．

```
unsigned short bayer[hsize][wsize]; // Bayerパターン配列
                                          画像データ
COLORREF color_d[hsize][wsize];      // RGBカラー配列構造体
FILE *fp;                            // RAWモード・ファイル・
                                        データの読み出し用
char *fname;
unsigned char *pgm;                  // ファイル読み出し
                                          作業領域
int width2 = 0xee0,height = 0xad8,intensity;
                         // S5200のwidth×2とheightの設定
int w_offset, h_offset;
char head[48];
unsigned char str[4]= {0xff, 0xff, 0xff, 0xff};
int buff_size = 24000;
int raw_data;
float cof = (float(0xff)/float(0x3fff));
        // 表示のための14ビット→8ビット化係数
int raw_entry, raw_addr;

pgm=(unsigned char *)malloc(buff_size);
      // ファイル読み出し作業のメモリ確保
fp = fopen("filename.RAF","rb");
      // ファイル・オープン: filename.pgmのバイナリ読み出し
fseek (fp, 0, SEEK_SET);
      // filename.RAFの先頭アドレス・セット
fread (head, 1, 48, fp);     // ファイル・ヘッダ部の読み出し
fseek (fp, 0x64, SEEK_SET);
      // RAWデータの先頭アドレスを取り出すためのアドレス・セット
fread (str, 1, 4, fp);
      // RAWデータの先頭アドレス取り出し(32ビット)
raw_entry = (str[0] << 24 | str[1] << 16 | str[2] << 8
                                                  |str[3]);
      // RAWデータの先頭アドレス・セット(32ビット)
w_offset = 0x370; // 切り出し幅オフセットのセット
h_offset = 0x470; // 切り出し高さオフセットのセット
raw_addr = raw_entry + h_offset*width2 + 2*w_offset;
                  // 切り出しの先頭アドレスをセット
fseek (fp, raw_addr, SEEK_SET);
                  // 切り出しの先頭アドレスへポインタ・セット
for (j = 0; j < hsize; j++){
fread (pgm, 1, width2, fp); // 1行分の切り出し
k = 0;
for (i = 0; i < wsize; i++){
if ( (i % 2 == 0) && (j % 2 == 0)){
    raw_data = (pgm[k+1] << 8 | pgm[k]);
                  // RAW画像データ1画素(16ビット)分取り出し
    bayer[j][i]=raw_data; // Bayer配列データへ格納
    d = raw_data*cof; // 表示のために8ビット化データ・セット
    k = k + 2;            // 次の画像データのアドレスへ
// オリジナルBayer配列データの色付け
{if (((i+j) % 4 == 0)) color_d[j-kk][i-kk] = RGB(d,0,0);
                                               // red表示
if ( ((i+j-2) % 4 == 0) ) color_d[j-kk][i-kk] = RGB
                                   (0,0,d);} // blue表示
}
else if ( (i % 2 == 1) && (j % 2 == 1)) {
raw_data = (pgm[k+1] << 8 | pgm[k]);
                  // RAW画像データ1画素(16ビット)分取り出し
bayer[j][i]=raw_data; // Bayer配列データへ格納
d = raw_data*cof;     // 表示のために8ビット化データ・セット
k = k + 2;            // 次の画像データのアドレスへ
color_d[j][i] = RGB(0,d,0);} // Green表示
else {color_d[j][i] = RGB(0,0,0);bayer[j][i]=0;}
                                    // 非画素部の黒表示
};
}
fclose(fp);  // ファイル・クローズ
for ( j = 0; j < hsize; j++){
for (i = 0; i < wsize; i++){
    pDC->SetPixel(i, j, color_d[j][i]);
                            // オリジナルBayer配列カラー表示
}
}
delt = 1.0/zoom; // 拡大率zoomの逆数セット
for ( j = 2; j < hsize - 2; j++){
    poy=j*delt;           // 補間点，y座標のセット
    vg = (poy+1.0)/2.0; // greenの補間点，y座標のセット
    lg =int(vg);          // greenの補間原点: y座標整数化
    vg = vg - lg;         // greenの補間原点: y座標小数部
for (i = 2; i < wsize ; i++){
    pox=i*delt;
    wg = (pox + 1.0)/2.0; // greenの補間点，x座標のセット
    kg =int(wg);            // greenの補間原点: x座標整数化
    wg = wg - kg;           // greenの補間原点: x座標小数部
    // 45 ° 回転操作
    poy=((j-i)*delt+4.0)/4.0; // 45 °回転y座標
    jl = floor(poy);          // 45 °回転y座標整数化
    vr = poy - jl;      // redの補間比: 45 °回転y座標小数部
    pox=((i+j)*delt)/4.0;     // 45 °回転x座標
    ik = floor(pox);          // 45 °回転x座標整数化
    wr =pox -ik;        // redの補間比: 45 °回転x座標小数部
    // 45 °回転の補間原点を求める
    kr = ik-jl+1; // red y座標
    lr = ik+jl-1; // red x座標

    poy=((j-i)*delt+2.0)/4.0; // 45 °回転y座標
    jl = floor(poy);          // 45 °回転y座標整数化
    vb = poy - jl;     // blueの補間比: 45 °回転y座標小数部
    pox=((i+j)*delt+2.0)/4.0; // 45 °回転x座標
    ik = floor(pox);          // 45 °回転x座標整数化
    wb =pox -ik;       // blueの補間比: 45 °回転x座標小数部
    // 45 °回転の補間原点を求める
    kb = ik-jl;   // blue y座標
    lb = ik+jl-1; // blue x座標
}
}
```

れの色ごとに可逆圧縮することです．拡大表示の場合と同じように，RとBは垂直・水平方向に並んでいるので問題ありませんが，Gは45 °回転方向に並んでいるのでこの点を考慮する必要があります．ここで考えられるのは，

- 45 °回転なしで水平方向に順番に並べていき，奇数行と偶数行をマージしてしまう
- 45 °回転なしで奇数行はそのまま水平方向に順番に並べていき，偶数行は補間した値を選んで，垂直方向の並びもそろえる
- 45 °回転なしで水平方向に順番に並べていき，奇数行と偶数行は分離する
- 45 °回転して並べる

といった方法があります．

Zhang氏は，Gの欠損している画素を周辺のGからの平均によって求めて，その位置のRとBの差を求めています．すなわち，次式で求めることになります．

$$\begin{cases} r(2i,\ 2j) = R(2i,\ 2j) - G(2i,\ 2j) \\ b(2i+1,\ 2j+1) = B(2i+1,\ 2j+1) - G(2i+1,\ 2j+1) \end{cases} \quad \cdots\cdots\cdots(21)$$

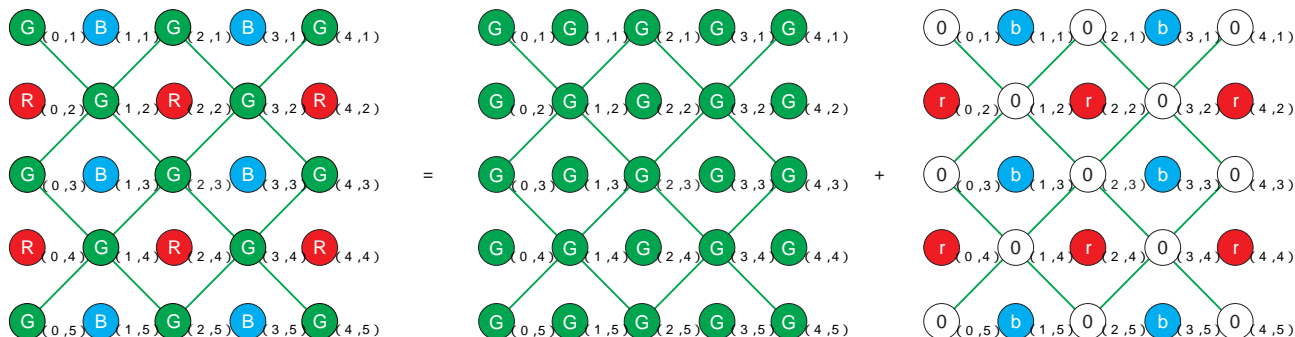



図19　Bayerパターン・カラー・フィルタ配列の分解

Bayerパターン・カラー・フィルタ配列は，G，r＝R－G，b＝B－Gに分解される．

（**a**）Bayerパターン・カラー・フィルタ配列のRAWデータ

LL	HL	LL	HL
LH	HH	LH	HH
LL	HL	LL	HL
LH	HH	LH	HH

（**b**）2次元Wavelet変換

図20　Bayerパターン・カラー・フィルタ配列と2次元Wavelet変換

図に示すように，Bayerパターン・カラー・フィルタ配列と2次元Wavelet変換は，両方とも2×2周期サンプリング・パターンをもっている．

図19に示すように，Bayerパターン・カラー・フィルタ配列は，G，r＝R－G，b＝B－Gに分解されます．

さらに興味深いのは，**図20**に示すように，Bayerパターン・カラー・フィルタ配列と2次元Wavelet変換は両方とも2×2周期サンプリング・パターンを持つことです．このことは，周波数と空間領域において2次元Wavelet変換がBayerパターン・カラー・フィルタ配列を効率的に表現することを意味します．Zhang氏はこのような理由で，2次元Wavelet変換による可逆画像圧縮がBayerパターン・カラー・フィルタ配列のRAWデータを直接，画像圧縮するのに適していると言っています．1画素当たり5ビット程度で表現できるようです．これ以上になると話がかなり専門的になるのでこれくらいにしておきますが，詳しくは参考文献(5)を参照してください．

参考・引用*文献

(1) 田丸雅也，小田和也，乾谷正史；新構造イメージセンサー「スーパーCCDハニカム」の原理と応用，FUJIFILM RESEARCH &DEVELOPMENT．No. 46-2001.
(2) Yamada, T. et al.；“A Progressive Scan CCD Image Sensor for DSC Applications”, IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 35, NO. 12, Dec. 2000.
(3) Lukin A. and Kubasov D；“High-Quality Algorithm for Bayer Pattern Interpolation”, Programming and Computer Software, pp.347-358, Translated from Programmirovanie, Vol. 30, No. 6, 2004.
http://graphics.cs.msu.su/en/publications/text/prog2004lk.pdf
(4) Kimmel R.；“Demosaicing: Image reconstruction from color CCD samples”, IEEE Trans. on Image Processing, 8(9):1221-8, Sep. 1999.
http://www.cs.technion.ac.il/~ron/pub.html
(5) Zhang N. and Wu X.；Lossless Compression of Color Mosaic Images, IEEE Trans. On Image Processing, Vol. 15, No. 6, pp.1379-1388, June 2006.
(6) Lian N-X., Chang L., Zagorodnov V. and Tan Y-P.；“Reversing Demosaicking and Compression in Color Filter Array Image Processing: Performance Analysis and Modeling”, IEEE Trans. On Image Processing, Vol. 15, No. 11, pp.3261-3278, Nov. 2006.
(7) 外村元伸；ディジタル画像処理のための各種補間法を理解する，Design Wave Magazine，2006年7月号，pp.78-84，CQ出版社．
(8) 外村元伸；キュービック・スプライン補間演算器の設計，Design Wave Magazine，2006年8月号，pp.120-126，CQ出版社．
(9) 外村元伸；拡大・縮小スケーリング・アルゴリズム演算の実装と評価，Design Wave Magazine，2006年9月号，pp.135-140，CQ出版社．

とのむら・もとのぶ
大日本印刷(株) 電子デバイス事業部 電子デバイス研究所

<筆者プロフィール>

外村元伸．今回の記事で使うRAWモード画像の写真を撮影するために，彼岸花やコスモスが咲いている場所を探しまわりました．遠くまで出かけて，家の近くまで帰ってきたところで彼岸花を2～3本見つけました．周囲を探すと，群生している場所がありました．